Learning geometric phase field representations

Yannick Kees

Born 14.05.1998 in Daun, Germany

28th October 2022

Master's Thesis Mathematics

Advisor: Prof. Dr. Martin Rumpf

Second Advisor: -

Institute for Numerical Simulations

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Danksagung

Mit dieser Seite möchte ich mich bei allen Personen bedanken, die auf unterschiedliche Art und Weise zum Gelingen dieser Arbeit beigetragen haben.

Contents

Introduction

1 Surface reconstruction problem

1.1 Storing data

Storing geometric data, such as 3D-objects can be very challenging. There are many ways of how one can do this, all with their own advantages and disadvantages. We want to explore some examples of them, based on [Pou20]. The most intuitive representations are the **explicit** representation. These are the ones, where from the definition we can directly obtain the precise location of the objects. There are four major types of them.

- **Point clouds.** Point clouds are unordered sets of points, where every point consists of different features, like its coordinates, the color at this point or the corresponding normal vector at this position. The big advantage of point clouds is, that this form of representation is closest to the raw data output that we would gain from sensors, like depth cameras or laser scanners. In these cases, the points are sampled from the surface of a geometrical object. For the sampling, we need no further assumption on how they are distributed on the surface. However, we run into problems if we want to visualize them. Rendering the bare point cloud is very easy, but often we are much more interested in reconstructing the original object. Since point clouds do not contain much information of the geometry or topology of the underlying objects, it they do not produce water tight surfaces right away. So for this, we might need to convert the point cloud into one of the other representation formats.

- **Voxeling.** If we want to represent data in the 2D-case, we often use images as representation. Images can be seen matrices, where every entry describes a same sized square, called pixel. Voxeling is the approach of doing this one dimension higher. Therefore, we consider now a three dimensional matrix, where every entry describes the properties of a certain cube. The big advantage of this representation is that it gives us a vector space structure, which allows us to perform mathematical operations on the data, like addition or convolution. However, a downside to this is, that we need to specify the resolution, so the size of the individual cubes. If we choose the size too small, due to the cubical growing of numbers of cubes, we will have to expect very high memory requirement, even if we are only dealing with spatial data. On the other hand, if we choose the cube size too big, we get too pixelated results, which can lead to a loss of information but also does not look very natural.

- **Meshing.** Another way of producing surfaces is by using meshes. We start by sampling a point cloud from the underlying object and then we connect some of the nearby points with edges. We do this until we have approximated the object with simple small geometric figures. One example for 3D-data is by using triangles. Finding a valid triangulation of points is quiet challenging. One of the most popular and most used methods for this problem is called Delaunay Triangulation ([Del34]), but this problem is still not fully solved. The difference to the voxeling approach is that meshes can only be used to represent the surfaces of objects but are not suitable for solid objects. But since the sizes of the different polygons in the mesh do not need to be the same, we get a better approximation, yielding in more natural looking visualization.

- **Image view.** The last approach can only be used for 3D data. The idea is that we take images from the object from many angles and therefore reducing the 3D data problem by one dimension to a 2D problem, accepting a loss of information. Benefits of this method are for example that the effect of noise and incompleteness is reduced and we do not need to deal with occlusion. However, the question is how many images are enough for a good representation.

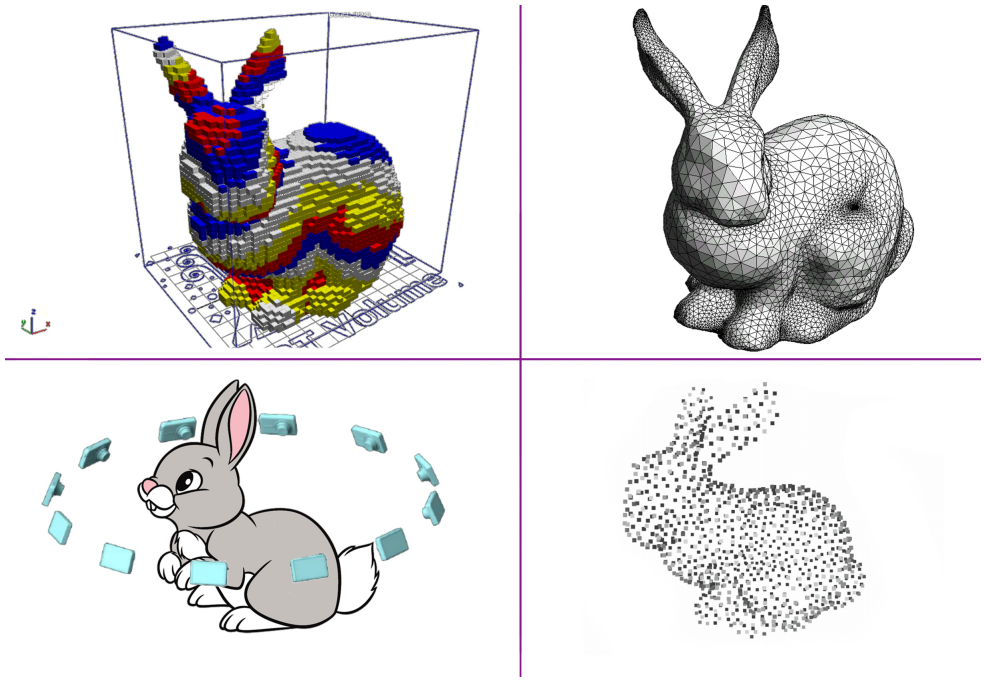


Figure 1: The four explicit ways of storing data

Instead of these representations, we will focus on a less visual representation for data, the **implicit representation**. For this, we will consider the given data as the isocontour of a scalar function. The question that we are dealing with here, is given data some data in a explicit representation, how can one find a function for the implicit representation.

1.2 Initial problem

Suppose we have given a space $\Omega \subseteq \mathbb{R}^d$ and a surface $\mathcal{S} \subseteq \Omega$ and from this surface we sample a point cloud $\mathcal{P} = \{p_1, \dots, p_n\}$. For now we will only consider the coordinates as features of the point cloud, i.e. X is a subset of Ω. Now our goal will be, given this point cloud, to reconstruct the original surface $\mathcal{S}$. In order to do this, we have to think about an easy way, to describe the set $\mathcal{S}$ properly. While finding a direct representation through some parametrisation is rather difficult, the easier way to use an implicit representation. In our case, we want to write the set as a zero-level set. That means, we want to find a function $u : \Omega \to [-1, 1]$, such that it holds

$$\mathcal{S} = \left\{ x \in \Omega \;\middle|\; u(x) = 0 \right\}.$$

Note, that this representation also gives us a disjoint division of our space Ω into three parts: our surface $\mathcal{S}$, the interior of the surface $\mathcal{I} = \left\{ x \in \Omega \;\middle|\; u(x) < 0 \right\}$ and the exterior $\mathcal{O} = \left\{ x \in \Omega \;\middle|\; u(x) > 0 \right\}$.

If u is a solution to the surface reconstruction problem, then we can easily see that $-u$ is also one. So the choice of interior and exterior is really arbitrary.

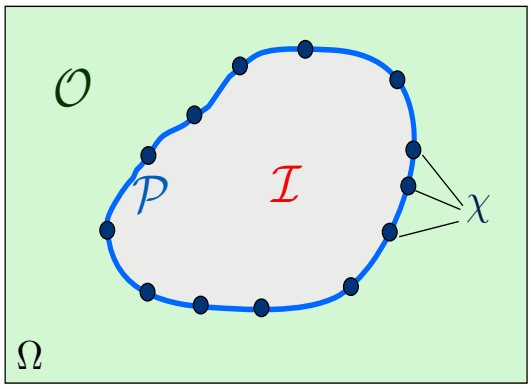


Figure 2: A 2D-example for the setting of surface reconstruction

For the function u, that we have used for the implicit representation, we only care about its zero-crossings and not so much about what happens outside of $\mathcal{S}$. That means for the outside behavior we obtain an additional degree of freedom. Based on that, we can consider three major classes of functions, that we could use for the implicit representation.

- **Indicator functions.** The easiest way of describing S is by the indicator function of its compliment $u_{IF} = \mathbb{1}_{S^c}$. This function is 0 if and only if a point lies on the surface and is 1 else wise. If we use this for the surface reconstruction problem, the interior will always be the empty set. That is why we will neglect this approach for now. But this also shows, that the problem is not uniquely solvable and therefore ill posed.

- **Occupancy functions.** These are all the functions that can be written as

$$u_{OF}(x) = \begin{cases} 1 & x \text{ is inside} \\ 0 & x \text{ is on the boundary} \\ -1 & x \text{ is outside} \end{cases}.$$

  The image of these functions is also discrete and they can be seen as a more general version of indicator functions.

- **Signed distance functions (SDF).** These functions also encode the distance between the given point and the original surface $\mathcal{S}$. If we have given an occupancy function u_{OF} we get a signed distance function u_{SDF} by

$$u_{SDF}(x) = u_{OF}(x) \cdot d(x, \mathcal{S})$$

  where $d(x, S)$ is the distance from the point x to S. So a signed distance functions maps every point to its positive distance to S, if the point is inside and to the negative distance if it is outside. They can be seen as a continuous version of the occupancy functions. Note that given a signed distance function it is very easy to compute an occupancy function, but the other way round is challenging, if we do not know S in advance.

1.3 Classical approach to surface reconstruction problem

We want to start by discussing some of the „classical“ approaches for solving the surface reconstruction problem, that are the ones that give a direct construction of u. The idea for them is always, that we consider a finitely generated vector space span $\{\phi_i \mid i = 1, \dots, n\}$ and want to write

$$u = \sum_{i=1}^{n} \alpha_i \phi_i \tag{1}$$

as a linear combination of the generating elements. We briefly want to take a look at two of them.

The first approach that we want to take a look at is from [CBC+01] using signed distance functions. Here we formulate the surface reconstruction problem as an interpolation problem. We do not only interpolate at the points of the point cloud, because then the function that is constant 0 would be a solution but we use some additional points that we obtain from going from the original points into the direction of the normal vector. The idea is that, if we make the steps small enough, we end up in the interior/exterior and know the distance from this point to the surface. So, for some $\varepsilon > 0$ want to interpolate the points

$$u(p_i) = 0 \qquad u(p_i + \varepsilon n_i) = \varepsilon \qquad u(p_i - \varepsilon n_i) = -\varepsilon \qquad \forall i = 1, \dots, n$$

where n_i is the outgoing normal vector in p_i, so the interpolant should represent a signed distance function. There are many ways of how one can solve interpolation problems. One approach is by using polynomials. The downside for them is, that if we use high order polynomials, the functions will heavily start oscillating in between the points. This behaviour is called Runge's phenomenon (see [Run01]). Another approach for interpolation problem is by using radial basis functions. These are all the functions, which values only depend on the distance of the input to the origin. One examples would be $\Phi(x) = \|x\|$. The problem with these type of function is that they may produce bad results in the extrapolation, that is the behavior of the function away from the points to be interpolated. In practice, we get the best results by using a mixture of radial basis functions and a low degree polynomial

$$u(x) = \sum_{i=1}^{n} \alpha_i x^i + \sum_{i=1}^{N} \beta_i \Phi\left(x - x_i\right), \tag{2}$$

where $\{x_i\} = \bigcup_i \{p_i, p_i \pm \varepsilon n_i\}$ are the points to be interpolated. In order to estimate the corresponding coefficients α_i and β_i, we insert (2) into the interpolation conditions and end up with a system of linear equations, that we then need just to solve.
This approach was later improved in [TO02], [HQD02] and more recently in [ZZW19]. Other ways of constructing signed distance functions can be found in [ZO02].

The second approach is from [MK06]. There, we want to solve the surface reconstruction problem via indicator function. The idea of this is, that the gradient of the indicator function is almost

everywhere zero, but on the surface. Therefore we start by extending the given normal vectors to a vector field $V : \mathbb{R}^3 \to \mathbb{R}^3$. Since we want the gradient of the function we are looking for to be similar to V, we consider the minimization problem

$$\min_u \int \|\nabla u(\xi) - V(\xi)\|^2 \, \mathrm{d}\xi \tag{3}$$

Using the Euler Lagrange formula, that we will talk about more later, one can show, that it is enough to solve $\Delta u = \nabla \cdot V$ (see **??**). We solve this partial differential equation using finite elements and the Galerkin method. Using (1), with ϕ being spline functions, we get the following linear system of equations, that we know how to solve

$$\sum_{j=1}^{n} \alpha_j \int \langle \nabla \phi_i(\xi), \nabla \phi_j(\xi) \rangle \, \mathrm{d}\xi = \int \langle \nabla \phi_i(\xi), V(\xi) \rangle \, \mathrm{d}\xi \qquad \forall j = 1, \ldots, n.$$

This was approach was later improved in [KH13] by adding a regularization term to the minimization problem

$$\min_u \int \|\nabla u(\xi) - V(\xi)\|^2 \, \mathrm{d}\xi + C \sum_{p \in \mathcal{P}} |u(p)|^2$$

that encourages the function to be zero in the original points.

$$\sum_{j=1}^{n} \alpha_j \left(\int \langle \nabla \phi_i(\xi), \nabla \phi_j(\xi) \rangle \, \mathrm{d}\xi + C \sum_{p \in \mathcal{P}} \phi_i(p) \phi_j(p) \right) = \int \langle \nabla \phi_i(\xi), V(\xi) \rangle \, \mathrm{d}\xi \quad \forall j = 1, \ldots, n.$$

One thing all that these attempts all have in common is, that for good results even on small point clouds, we will need to know normal vectors in advance and always end up with a large system of linear equations.
Another famous approach for solving the surface reconstruction problem with indicator functions is from [PA07], where they use the Voronoi diagram of the input points to approximate the normals of the surface. The approach we want to focus on is very different. Instead of giving a explicit construction of the function, we want to „learn“ it using artificial intelligence.

1.4 Deep neuronal networks

Before we explain, how we want to learn occupancy functions or signed distance functions, we want to start with a broad overview on artificial intelligence based on the first chapter in [Agg18]. In general **Artificial Intelligence** is a subsection of computer science and mathematics. It deals with machines mimicking functions typically associated with human cognition. We consider it as the attempt of a computer to recreate certain decision-making structures of the human brain. The techniques and processes that are used for solving these problems are summarized under the term **machine learning**. Instead of solving a problem explicit, we put all the data in a generic algorithm, that then can solve the problem. So machine learning analyses data to „learn“and then use this knowledge to make decisions. We differ between three categories, based on how the machine learning algorithm performs the learning.

(i) **Supervised Learning.** The algorithm should approximate a mapping function, that pre-

dicts the output for a given observation. We train the algorithm with with data, from which we already know the desired outputs. Output variables of the training data are often called labels or categories. It is common for classification models to predict a continuous value as the probability of a given example belonging to each output class The probabilities can be interpreted as the likelihood or confidence of a given example belonging to each class. A predicted probability can be converted into a category by selecting the class label, that has the highest probability. The algorithm should adjust itself in such a way, that the error between the predicted output and the correct labels gets minimized.
For example an email of text can be classified to be either „spam“ or „no spam“. So we train the algorithm by feeding it with E-Mails from which we already now weather they are spam or not and in the end the program should get a new mail and make a decision on its on.

(ii) **Unsupervised Learning.** These are the algorithms, that we supply with inputs but no prior known outputs. The algorithm should adjust itself in such a way, that similar inputs cause similar outputs. It identifies the patterns and differences in the input without any external assistance. Examples for this are clustering problems or detecting data anomalies.

(iii) **Reinforcement Learning.** These are the algorithms that work under the „trial-and-error“ principle. That means we consider an algorithm, that can perform several actions and is given constant feedback on its behavior in form of rewards. Based on that, the algorithm tries to find the best strategy for maximizing its reward.
This type of learning based on several principles in biology. For example, if a dog trainer is teaching his animal something, he will use rewards, whenever the dog does something that he wants him to do. Based in this positive reinforcement, the dog learns the commands of the trainer.

Since the late 2000s **deep learning** has been the most successful approach to supervised learning problems. It uses **artificial neuronal networks** to solve the problems we stated before. Any neuronal network always consists out of same three parts: the inputs form an input layer, the middle layers which perform the processing of the inputs are called hidden layers and the output forms the output layer. The hidden layers consist of **neurons**, that perform mathematical operation on their associated inputs. Each input to a neuron is associated with a different weight, which affects the computation. So neuronal networks represent functions of their given input by propagating the computed values from one layer to the next until they reach the final output layer. The key to neuronal networks is to adjust the weights in such a way, that the function represented by it solves the initial problem. In general we can think of artificial neuronal networks universal function approximaters.
We now consider the easiest case of an ANN, a **multi-layer perceptron (MLP)**. MLPs are the most widely used architecture for neuronal networks. To understand how they work we take a look at what happens in the neurons of the MLP. The neurons do two important things. First they calculate a weighted sum of their incoming data by performing an affine linear transformation to it. Afterwards the transformation is inserted into an **activation function** σ. A common choice for an activation function is the **Rectified Linear Unit (ReLU)** $\sigma(x) = \max\{0, x\}$. This function is the identity for all positive inputs and maps all negative values to 0. So if we

consider a single neuron, that gets n inputs x_i and is associated with weights w_i, the output of this neuron would be

$$\sigma\left(\sum_{i=1}^{n} w_i x_i + w_{n+1}\right).$$

For evaluating a MLP, we repeat this calculation for every neuron until we reach the output layer.

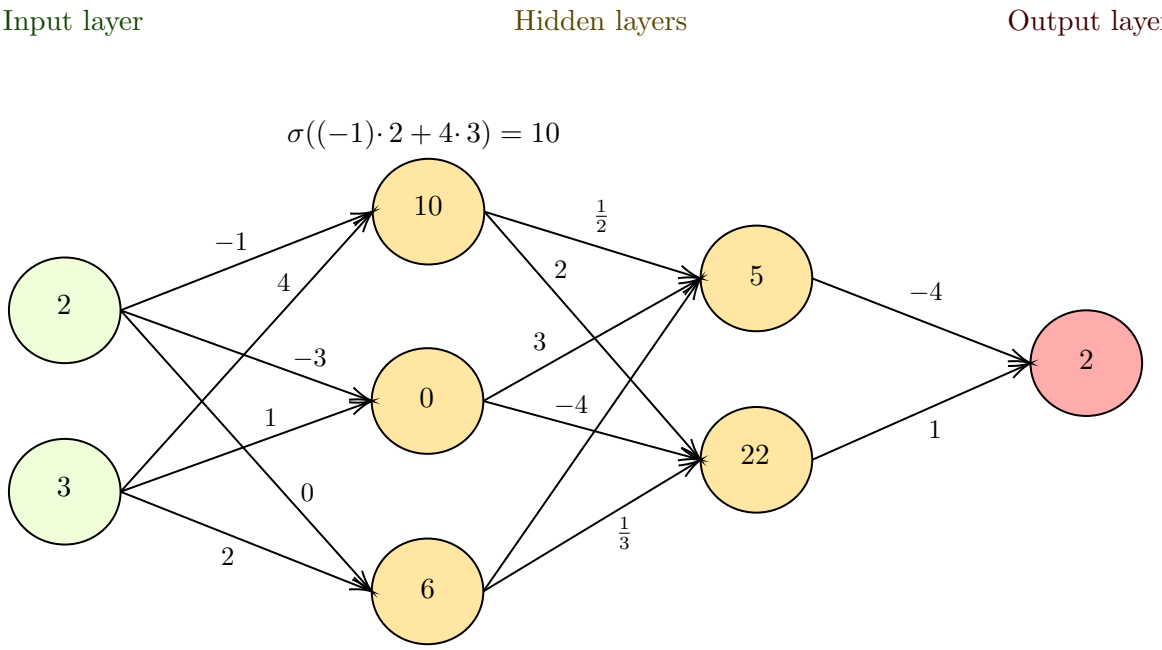


Figure 3: Example of the forward propagation of a neuronal network with two hidden layers, ReLU activation function and no bias

Now we want to use the MLP to solve supervised learning problems. Let f the by function representing the architecture of the network, $\vec{w}$ be the weights of the network summarized as a vector, $(\vec{x}_i)_{i=1}^n$ be the training data and $(\vec{y}_i)_{i=1}^n$ be the corresponding training data. For the training of the MLP, we need to measure how good the prediction are, the network is currently making. This means we need a way to measure the overall error of it. Now we can choose a **loss functional** $\mathcal{L}$, that is a map, that takes the predicted labels and the actual labels as input and returns the total error. The choice of the loss functional is crucial in what exactly the network should learn. Then we can write the supervised learning problem as the following minimization problem.

$$\min_{\vec{w}} \mathcal{L}\big((f(\vec{x}_i, \vec{w}))_{i=1}^n , (\vec{y}_i)_{i=1}^n \big)$$

For solving minimization problems there are plenty of numerical algorithms, that can be used, like gradient descent. That means we have reformulated the supervised learning problem into a problem, that we can handle and therfore found a solution to the original problem.
We now want to specify more, what a solution actually is. By increasing the number of layer and neurons, the network is able to learn more complex functions on the training set. That means if we have a network that is too small, we will not be able to get good results. However, making the network to big, will provide good performance on the training data, but does not guarantee good performance on unseen test data. This problem is called **overfitting** and arises

if the network learns a function, that is too specifically matched to the underlying training data.

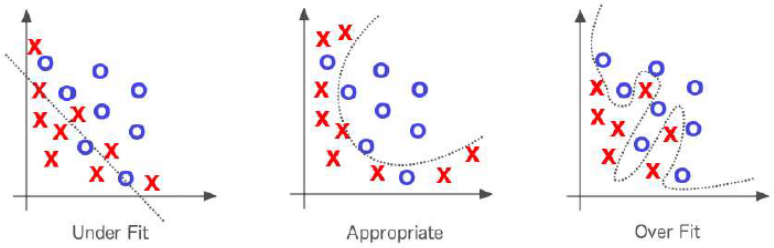


Figure 4: 2D example of under/over-fitting

One way of reducing the overfitting error is by adding a regularization term to the loss functional

$$\mathcal{L} = \mathcal{L}_{data} + \mathcal{L}_{reg}.$$

Here $\mathcal{L}_{data}$ is the loss between the predicted labels and the actual labels, and $\mathcal{L}_{reg}$ is function measuring the complexity of the network. So a appropriate solution function should map the inputs close to their labels, but should also be as simple as possible.

1.5 Learning implicit surface representations

Now we want to get back to the surface reconstruction problem. The idea is to consider it as a supervised learning problem, where the training data are the points of the point cloud, since we know, how our function should behave there. That means the training data are the points $x_i \in X$ and the corresponding labels are $y_i = 0$.
As the neuronal network f for the implicit representation of $\mathcal{S}$ we choose a MLP. The network should take a single point as input, therefore we need d inputs for the input layer. The output layer should be one neuron, that scores between 1 and -1. By the previous chapter, the only thing we need to specify now is how to choose the loss functional. That means we can reformulate the surface reconstruction problem as:

> Find a loss functional $\mathcal{L}$ such that $S = \{x \in \Omega \mid f(x, w_*) = 0\}$ with $w_* = \arg\min_w \mathcal{L}\big((f(p,w))_{p\in\mathcal{P}}, \mathcal{P}\big)$.

Next we want to get rid of the weights in the problem formulation. Therefore we need the following result for multi-layer perceptrons.

Theorem 1.1 (Universal approximation theorem):
Let $\Omega \subset \mathbb{R}^d$ be compact. For any map $f : \Omega \to \mathbb{R}$ continuous, there is a sequence of functions $(f_n)_{n\in\mathbb{N}}$ representing a MLP with ReLU activation functions and $d + 2$ hidden layers, such that

$$\lim_{n\to\infty} \|f - f_n\|_{C_0(\Omega)} = 0$$

You can find a proof of this theorem in [Han19]. The big takeaway from this is, that we can approximate any function with a neuronal network. For our loss function, this mean we no longer need to minimize over some weights, but rather consider a minimization problem over the whole space of functions. That means we solve:

Find a loss functional $\mathcal{L}$ such that $S = \{x \in \Omega \mid f_*(x) = 0\}$ with $f_* = \arg\min_f \mathcal{L}\big(f, \mathcal{P}\big)$.

Since we will need large networks for this, we need to address the overfitting problem. So let $\mathcal{L} = \mathcal{L}_{data} + \mathcal{L}_{reg}$ as before. The data loss is responsible for making the function 0 along $\mathcal{S}$. The regularization error is responsible for making the function as smooth as possible elsewhere.
In this thesis we are going to discuss several losses

1.6 Learning shape spaces

2 Mathematical prerequisites

2.1 Gamma convergence

In the context of machine learning our goal is always to find a minimizer for a given functional. These minimizers may be very hard to compute, so instead we want to find an approximation of this functional, that converges against it. The question is, what definition of convergence we use here. Since we are mainly interested in the minimizers of the functional, ideally we would want that minimizers of the approximation to converge to a minimizer of the original functional. One way of enforcing this, is to consider uniform convergence, but we will see in practice, that this assumption is often too strong.
So we need a new definition on what convergence of functionals should actually mean. This new definition was first developed by the italian mathematican Ennio de Giorgi in [Gio75]. We use the defintions and proofs from [Mut10].

Definition 2.1:
Given a metric space (X, d) and a family of functionals $\mathcal{F}_n \to \overline{\mathbb{R}}$, we say $\mathcal{F}_n$ **$\Gamma-$converges** to a functional $\mathcal{F} : X \to \overline{\mathbb{R}}$, if for every $x \in X$ the following two properties hold:

(i) For every sequence $(x_n)_{n\in\mathbb{N}} \subset X$ with $x_n \longrightarrow x$ it holds

$$\mathcal{F}(x) \leq \liminf_{n\to\infty} \mathcal{F}_n(x_n) \tag{4}$$

(ii) There exists one sequence $(\tilde{x}_n)_{n\in\mathbb{N}} \subset X$ with $\tilde{x}_n \longrightarrow x$ and

$$\limsup_{n\to\infty} \mathcal{F}_n(\tilde{x}_n) \leq \mathcal{F}(x)$$

We denote the convergence as $\mathcal{F}_n \xrightarrow{\Gamma} \mathcal{F}$.

The first property in the Γ-convergence definition is often called the lim inf-property and the second one lim sup-property. The sequence we used in the lim sup property is called recovery sequence. With this new definition we can now investigate how minimizers behave under this convergence.

Lemma 2.2:
Let $\mathcal{F}_n : X \to \overline{\mathbb{R}}$ be a family of functions and let x_n be a minimizer of $\mathcal{F}_n$. Suppose that $(x_n)_{n\in\mathbb{N}}$ converges to x and $\mathcal{F}_n \xrightarrow{\Gamma} F$. Then x is a minimizer of $\mathcal{F}$.

Proof: Let $y \in X$ arbitrary. By the second property of Γ-convergence we can find a sequence (y_n) with $y_n \longrightarrow y$ and $\limsup_{n\to\infty} \mathcal{F}_n(y_n) \leq \mathcal{F}(y)$. We obtain

$$\mathcal{F}(x) \overset{(4)}{\leq} \liminf_{n\to\infty} \mathcal{F}_n(x_n) \leq \limsup_{n\to\infty} \mathcal{F}_n(x_n) \leq \limsup_{n\to\infty} \mathcal{F}_n(y_n) \leq F(y)$$

Since y was arbitrary, we can take the minimum and get $\mathcal{F}(x) \leq \min_{y\in X} \mathcal{F}(y)$.

□

So if we have a sequence of functionals $\mathcal{F}_\varepsilon$ with corresponding minimizers u_ε and we want to find a minimizer for $\varepsilon \to 0$ we need two things: First the $\mathcal{F}_\varepsilon$ should Γ-converge and second the u_ε need to converge in the usual sense. Showing that the sequence of minimizers does converge can be really challenging, since we do not actually want to compute them. That is why, we want to find some additional properties of the functionals, under which we can drop this condition.

Definition 2.3:
A sequence of functionals $F_\varepsilon : X \to \overline{\mathbb{R}}$ is called **equi-coercive** if for every $t \in \mathbb{R}$ we can find a compact set $K_t \subset X$ such that

$$\{x \in X \mid F_\varepsilon(x) \leq t\} \subset K_t.$$

Theorem 2.4:
Let $\{\mathcal{F}_n\}$ be a equi-coervice family of functionals and let $\mathcal{F}_n \xrightarrow{\Gamma} F$, then the minimizers of $\mathcal{F}_n$ converge and their limit is a minimizer of $\mathcal{F}$.

2.2 Euler-Lagrange formula

Now that we have derived a convergence theory for minimizer of functionals, we want to focus more on how we can actually compute a minimizer. Functionals do not get scalar values as input, but functions. Still, there is a similar analogy, that is the minima of a functional are characterized by a certain partial differential equation. Want want to derive this PDE by reconsidering the scalar valued case using the following definition. We do so, based on the second chapter in [Arn20].
For differentiable scalar functions, that are defined on a subset $\mathbb{R}^d$, we can find candidates for local extrema, by checking where the derivative equals zero. More general, we want that the derivative in every direction is zero. To extend this on functionals, we need the following definition.

Definition 2.5:
Let $h_0 > 0$ and $\mathcal{F} : X \to \mathbb{R}$, where X is subset of an $\mathbb{R}$-vector space. Then we define the **first variation** of $\mathcal{F}$ in u in direction $\xi \in \left\{\tilde{\xi} \;\middle|\; u + h\tilde{\xi} \in X,\ \forall |h| < h_0\right\}$ as

$$\delta\mathcal{F}(u, \xi) := \psi'_{\mathcal{F},u,\xi}(0)$$

with $\psi_{\mathcal{F},u,\xi}(h) := \mathcal{F}(u + h\xi)$ for all $|h| < |h_0|$.

If $\mathcal{F}$ attains a local extremum in u_0, then $\psi_{\mathcal{F},u,\xi}$ has a local extremum in 0. Therefore, $\psi_{\mathcal{F},u,\xi}(h) = 0$ is a necessary condition for being in a local extrema. Note that in the scalar valued case, that means for $\mathcal{F} \in C^1(\Omega, \mathbb{R})$ we have

$$0 = \delta\mathcal{F}(u, \xi) = \langle \nabla\mathcal{F}(u), \xi \rangle$$

for any valid direction ξ and therefore can conclude $\nabla\mathcal{F}(u) = 0$.
Now we want to calcuate the first variation for functionals. We only consider **variational in-**

tegrals as functionals, that are the ones, that can be written as

$$\mathcal{F}(u) = \int_\Omega F(x, u(x), \nabla u(x))\,\mathrm{d}x$$

for some function $F \in C^1(V, \mathbb{R})$ with $V \subset \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d$. First, we need to make sure, that the first variation even exists.

Lemma 2.6:
Let $\Omega \subset \mathbb{R}^d$ open and $X := C^1(\overline{\Omega})$. Then for all $u_0 \in U := \left\{u \in X \;\middle|\; \left\{x, u(x), \nabla u(x) \;\middle|\; t \in \overline{\Omega}\right\} \subseteq V\right\}$ and every $\xi \in X$, the first variation of $\mathcal{F}$ in u_0 in direction ξ exists.

Proof:

□

To simplify the notation the following main theorem we set

$$\alpha := x \qquad \beta := u(x) \qquad \gamma := \nabla u(x)$$

so that we regard $\mathcal{F}(u) = \int_\Omega F(\alpha, \beta, \gamma)$.

Theorem 2.7 (Euler-Lagrange formula):
Let $u \in C^2(\Omega) \cap C^1(\overline{\Omega})$ be a local minimum of $\mathcal{F}$ in U with $\frac{\partial F}{\partial \gamma_i} \in C^1(V)$ for $i = 1, \ldots, d$, that is $\delta F(u, \xi) = 0$ for all $\xi \in C_0^\infty(\Omega)$. Then u fulfills the following partial differential equation

$$\frac{\partial F}{\partial \beta} - \nabla \cdot \left(\frac{\mathrm{d}F}{\mathrm{d}\gamma}\right) = 0.$$

Proof: We start by explicitly calculating the first variation.

$$\begin{aligned}
\delta\mathcal{F}(u, \xi) &= \frac{\mathrm{d}}{\mathrm{d}h}\mathcal{F}(u + h\xi)\bigg|_{h=0} \\
&= \frac{\mathrm{d}}{\mathrm{d}h}\int_\Omega F\left(x, u(x) + h\xi(x), \nabla u(x) + h\xi(x)\right)\,\mathrm{d}x\bigg|_{h=0} \\
&= \int_\Omega \frac{\mathrm{d}}{\mathrm{d}\beta}F\left(x, u(x) + h\xi(x), \nabla u(x) + h\xi(x)\right)\xi(t) \\
&\qquad + \sum_{i=1}^d \frac{\mathrm{d}}{\mathrm{d}\gamma_i}F\left(x, u(x) + h\xi(x), \nabla u(x) + h\xi(x)\right)\frac{\partial \xi}{\partial x_i}(x)\,\mathrm{d}x\bigg|_{h=0} \\
&= \int_\Omega \frac{\mathrm{d}}{\mathrm{d}\beta}F\left(x, u(x), \nabla u(x)\right)\xi(t) + \sum_{i=1}^d \frac{\mathrm{d}}{\mathrm{d}\gamma_i}F\left(x, u(x), \nabla u(x)\right)\frac{\partial \xi}{\partial x_i}(x)\,\mathrm{d}x
\end{aligned}$$

Where we used the dominated convergence theorem in the third line. Then using the product

rule for derivatives and the Gaussian formula we obtain from the previous formula

$$\begin{aligned}
0 &= \delta\mathcal{F}(u,\xi) \\
&= \int_\Omega \frac{\mathrm{dF}}{\mathrm{d}\beta}\xi + \sum_{i=1}^{d} \frac{\mathrm{dF}}{\mathrm{d}\gamma_i}\frac{\partial \xi}{\partial x_i}\,\mathrm{d}x \\
&= \int_\Omega \frac{\mathrm{dF}}{\mathrm{d}\beta}\xi + \sum_{i=1}^{d} \left(\frac{\partial}{\partial x_i}\left(\xi\frac{\partial F}{\partial \gamma_i}\right) - \xi\frac{\partial}{\partial x_i}\frac{\partial F}{\partial \gamma_i}\right)\,\mathrm{d}x \\
&= \int_\Omega \left(\frac{\mathrm{dF}}{\mathrm{d}\beta} - \nabla\cdot\left(\frac{\partial F}{\partial \gamma}\right)\right)\xi + \nabla\cdot\left(\xi\frac{\partial F}{\partial \gamma}\right)\,\mathrm{d}x \\
&= \int_\Omega \left(\frac{\mathrm{dF}}{\mathrm{d}\beta} - \nabla\cdot\left(\frac{\partial F}{\partial \gamma}\right)\right)\xi\,\mathrm{d}x + \int_{\partial\Omega}\left\langle \underbrace{\xi(x)}_{=0}\frac{\partial F(x)}{\partial\gamma}, n(x)\right\rangle \mathrm{d}\mathcal{H}^{d-1}(x) \\
&= \int_\Omega \left(\frac{\mathrm{dF}}{\mathrm{d}\beta} - \nabla\cdot\left(\frac{\partial F}{\partial \gamma}\right)\right)\xi\,\mathrm{d}x
\end{aligned}$$

Since $\xi \in C_0^\infty$ was arbitrary, the result follows from the fundamental theorem of calculus.

□

We want to consider again the example from (**??**). Here in the variational setting $\min_u \int \|\nabla u(\xi) - V(\xi)\|^2\,\mathrm{d}\xi$, the integrand is given by

$$F(\alpha,\beta,\gamma) = \|\gamma - V\|^2$$

Applying the Euler-Lagrange formula yields that the minimizer u_0 fulfills

$$\begin{aligned}
&\frac{\partial}{\partial\beta}\|\gamma - V\|^2 - \nabla\cdot\left(\frac{\mathrm{d}F}{\mathrm{d}\gamma}\|\gamma - V\|^2\right) = 0 \\
\Rightarrow\quad & \nabla\cdot 2(\gamma - V) = 0 \\
\Rightarrow\quad & \Delta u = \nabla\cdot\nabla u = \nabla\cdot\gamma = \nabla\cdot V
\end{aligned} \tag{5}$$

2.3 Bounded variation and surface perimeter

[Giu84]

Definition 2.8:
Let $\Omega \subset \mathbb{R}^d$ be open and $f \in L^1(\Omega)$. We define the **total variation** of f as

$$\int_\Omega |Df| := \sup_{\substack{g\in C_0^1(\Omega,\mathbb{R}^d)\\ \|g\|_\infty \le 1}} \int_\Omega f(x)\mathrm{div}(g)(x)\,\mathrm{d}x$$

If $\int_\Omega |Df| < \infty$ we say f has bounded variation and we denote the space of all functions with

bounded variation as $BV(\Omega)$.

Note that $\int_\Omega |Df|$ is just some notation and does not actually need that f is differentiable, but can be applied to a much bigger group of functions. Yet, this notion we use suggest that the variation can be seen as some form of integration over the derivative. In fact, the next lemma will show that if f is continuous differentiable, then this is actually holds.

Lemma 2.9:
If $f \in C^1(\Omega)$ then the total variation is the integral over the absolute value of the gradient

$$\int_\Omega |Df| = \int_\Omega |\nabla f(x)| \, \mathrm{d}x.$$

Proof: In order to prove this, we first apply integration by parts to obtain the following expression.

$$\int_\Omega |Df| := \sup_{\substack{g \in C_0^1(\Omega, \mathbb{R}^d) \\ \|g\|_\infty \leq 1}} \int_\Omega f(x) \mathrm{div}(g)(x) \, \mathrm{d}x = - \sup_{\substack{g \in C_0^1(\Omega, \mathbb{R}^d) \\ \|g\|_\infty \leq 1}} \int_\Omega \sum_{i=1}^d \frac{\partial f}{\partial x_i} g(x)_i \, \mathrm{d}x$$

Since g is bounded by 1, we get

$$\int_\Omega |Df| \leq \int_\Omega |\nabla f(x)| \, \mathrm{d}x.$$

For the other inequality we choose as test function $g = \frac{\nabla f}{|\nabla f|}$. This is bounded by 1 and we obtain

$$\int_\Omega |Df| \geq \int_\Omega |\nabla f(x)| \, \mathrm{d}x,$$

which yields the assertion.

□

If we go back to the surface recontruction problem, our goal was to find a functional, that would minimize the boundary of some domain. If a boundary is smooth enough, we already know that we can measure its size with the correct dimensional Hausdorff measure. If not, the concept of total variation allows us now to define the size of them, by looking at the total variation of the corresponding indicator function.

Definition 2.10:
Let $I \subset \Omega$. Then we define the **surface perimeter** by

$$\mathrm{per}_\Omega(I) := \int_\Omega |D\mathbb{1}_I|$$

The following lemma show, why we can really see the surface perimeter as a more general version of the measure of the boundary.

Lemma 2.11:
Suppose $I \subset \Omega$ has a C^2-boundary, then we can write the perimeter as

$$\mathrm{per}_I(\Omega) = \mathcal{H}^{d-1}(\partial I).$$

Proof: Let $g \in C_0^1(\Omega)$ with $\|g\|_\infty \leq 1$. Then we can apply the Gauss-theorem and with $n : \partial I \to \mathbb{R}$ being the function, that maps a vector to its outgoing normal, we obtain

$$\mathrm{per}_I(\Omega) = \int_\Omega \mathbb{1}_I \mathrm{div}(g)(x)\,\mathrm{d}x = \int_I \mathrm{div}(g)(x)\,\mathrm{d}x = \int_{\partial I} \langle g(x), n(x)\rangle\,\mathrm{d}x$$

And now with the real version of Cauchy-Schwarz we get

$$\int_\Omega \mathbb{1}_I \mathrm{div}(g)(x)\,\mathrm{d}x \leq \int_{\partial I} \underbrace{|g(x)| \cdot |n(x)|}_{=1\cdot 1}\,\mathrm{d}\mathcal{H}^{d-1}(x) = \mathcal{H}^{d-1}(\partial I)$$

For the other direction let $N : C^1(\mathbb{R}^n, \mathbb{R}^n)$ be an extension of the outer normal n. Then we consider all functions g that are of the form $g = \phi N$ for a test function $\phi \in C_0^\infty(\Omega)$ with $\|\phi\|_\infty \leq 1$. Then

$$\mathrm{per}_I(\Omega) = \sup_{\substack{g \in C_0^1(\Omega, \mathbb{R}^d) \\ \|g\|_\infty \leq 1}} \int_{\partial I} \langle g(x), n(x)\rangle\,\mathrm{d}\mathcal{H}^{d-1}(x) \geq \sup_{\substack{\phi \in C_0^\infty(\Omega, \mathbb{R}^d) \\ \|\phi\|_\infty \leq 1}} \int_{\partial I} \phi(x) \underbrace{\langle N(x), n(x)\rangle}_{=1}\,\mathrm{d}\mathcal{H}^{d-1}(x) = \mathcal{H}^{d-1}(\partial I)$$

□

3 Finding Loss functionals

3.1 Phase transition problem

Before we continue on the surface reconstruction problem, we want to consider a very related problem from physics: the phase transition problem. In this setting, we have a space Ω and in this we put two different liquids. We want to find out, how these liquids behave, therefore we denote the density of the two liquids by a function $u : \Omega \to [-1, 1]$. For some position $x \in \Omega$ the case $u(x) = 1$ means that in this position we have the first liquid and $u(x) = -1$ means we have only the second liquid. The case $u(x) = 0$ means we have a perfect mixing ratio of the two liquids in this point. What we will see in reality is that the both fluids will arrange themselves in such a way, that the intersection among them has minimal area. That means, we want that almost every point within Ω should be either 1 or -1. To formulate this more mathematically, we need the following definition.

Definition 3.1:
A **double-well potential** W is a continuous function $W : \mathbb{R} \to [0, \infty)$, such that

(i) $W(x) = 0$ if and only if $x \in \{1, -1\}$.

(ii) There exist $L, R > 0$ such that $W(z) > L|z|$ for every $|z| \geq R$.

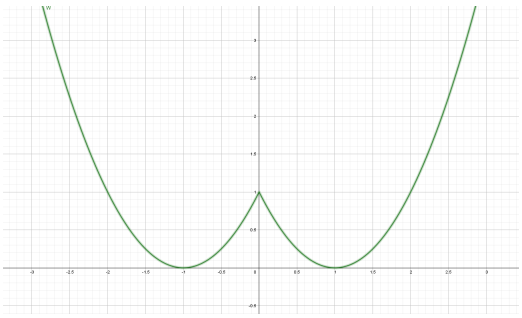


Figure 5: Graph of a double-well potential

Let W be a double-well potential. With the reasoning from before, we want $W(u)$ to be zero as much as possible. More mathematically speaking, we are looking for a solution for the following minimization problem:

Phase transition

$$\min_{u \in L^1(\Omega, [-1,1])} \int_\Omega W(u(x))\, \mathrm{d}m(x)$$

$$\text{s.t.:} \quad \int_\Omega u(x)\, \mathrm{d}m(x) = m$$

The secondary condition here comes from the fact, that we input a certain fixed amount of the two liquids, so m is a constant value.

There is one problem with the minimization problem stated above and that is, it does not minimize the area of interference of the liquids. In fact, every decomposition of the space Ω, where u is 1 on the one part and -1 in the other part, is a minimum for this functional. So we want to change the minimizer in such a way, that there is really only one minimum. We do so by adding a regularization term to it.

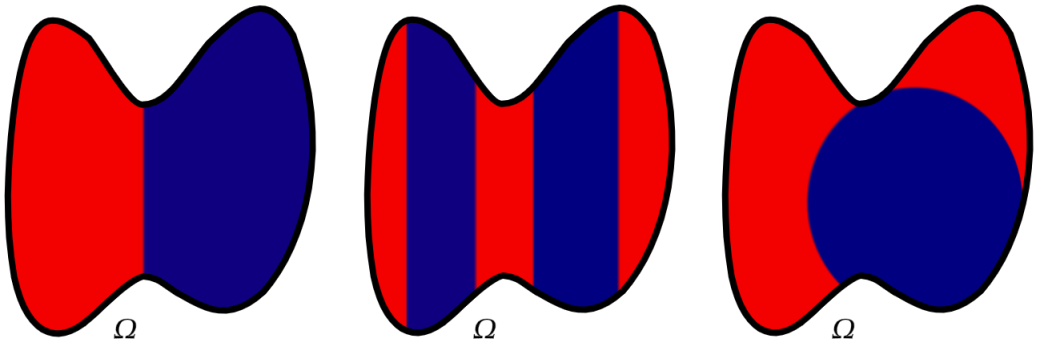


Figure 6: All these images are local minima of the phase transition energy, but we want the left one to be the only minimum

Adding a regularization term here does the same as adding a regularizer term in the context of machine learning, that is getting rid of over fitting. Now in our case we decide to use a **Dirichlet regularizer**, which is just the integral over the absolute value of the gradient. In order to ensure existence, we minimize over the Sobolov space H^2. This means we end up with the following minimization problem, that was found by Modica/Mortola.

Modica-Mortola

$$\min_{u \in H^2(\Omega,[-1,1])} \mathcal{M}_\varepsilon(u) := \int_\Omega \frac{1}{\varepsilon} W(u(x)) + \varepsilon \|\nabla u(x)\|^2 \, \mathrm{d}m(x)$$

$$\text{s.t.:} \qquad \int_\Omega u(x) \, \mathrm{d}m(x) = m$$

We now actually get a whole family of minimization problems, that depend on a single parameter $\varepsilon > 0$. We can see, that as ε decreases, the contribution of the double well potential increases, while the one of the regularizer term decreases.

Theorem 3.2 (Modica-Mortola):
Under some assumptions on Ω, there exists a functional $\mathcal{M} : \ H^2(\Omega, [-1,1]) \to \mathbb{R} \cup \{\infty\}$, such that $\mathcal{M}_\varepsilon$ is Γ-converging to $\mathcal{M}$, where

$$\mathcal{M}(u) := \begin{cases} c|Du|(\Omega) & u \in BV(\Omega, \{1,-1\}) \\ \infty & \text{else} \end{cases}$$

3.2 Modica Mortola for Surface reconstruction

This chapter and the next one are summarizing [Lip21]. Now we want to use the approach we derived for the phase transition problem onto the surface reconstruction problem. We already know, that there are multiple ways to address this problem. There has been a lot of research done, trying to create loss functionals for either an occupancy function or a signed distance function. But for now we want to use the approach by Yaron Lipman. In this paper he proposed the idea of learning first an occupancy function and from this create a signed distance function. So the final loss he uses to create a solution, manages to combine features of both of these losses. But before we do this, we want to state, what a solution in this case actually is. Ideally the minimizer u_0 of the loss functional should fulfill the following properties.

(i) **Proper occupancy.** We want to solve the surface reconstruction problem by using an occupancy function, so the image of u_0 should be 1 or -1 almost everywhere. To say this differently we want $u_0 \in BV(\Omega, \{\pm 1\})$.

(ii) **Zero reconstruction loss.** The original point cloud should be in the set of zero crossings of u_0, that is $u_0(x) = 0$ for all $x \in X$.

(iii) **Minimal perimeter.** If we would just demand the first two properties to hold, every arbitrary decomposition of Ω into an exterior and interior would be a solution. From all these decompositions, we want that functions, that has a minimal perimeter. So u_* should minimise $\mathrm{per}_\Omega(\mathcal{I})$.

(iv) **Easy way of getting a SDF.** Since the representation of a occupancy function can be challenging, we also want a way, that given the occupancy function, produces a signed distance function.

If we apply the approach of the surface reconstruction problem, we can regard the first liquid as the interior of the surface and the second liquid as the exterior. Using then the Modica-Mortola functional would already ensure the proper occupancy and minimal perimeter property. Since we do not have a mass constraint we replace it by the zero reconstruction loss, by summing over the absolute values averaging integral over small balls of the original point in the point cloud. We also do not treat this as a secondary condition, but as a summand in the functional.

PHASE$_0$

$$\min_{u \in H^2(\Omega,[-1,1])} \mathcal{F}_\varepsilon(u) := \int_\Omega \frac{1}{\varepsilon} W(u(\xi)) + \varepsilon \|\nabla u(\xi)\|^2 \, \mathrm{d}m(\xi) + \frac{\lambda}{|X|} \sum_{p \in \mathcal{P}} \left| \fint_{B_\delta(p)} u(\xi) \, \mathrm{d}m(\xi) \right|$$

Here λ is some positive constant, depending only on ε, that we specify later. By minimizing the functional, we want to enforce the averaging integral to be 0 around these small balls of radius δ. Note, that this is a stronger condition, that just wanting the function to be zero in the underlying points from the point cloud, because the integral becomes only zero, if some parts of it cancel out. This should force the function to change its sign along the surface S.

Figure 7: Minimizing the integral over some ball around a point $x \in X$ yields in a change of sign of the function u.

Now similar to the Modica-Mortola theorem, we can prove Gamma convergence of the sequence of functionals.

Theorem 3.3 (Lipman):
If we choose $\lambda = \lambda(\varepsilon)$, in such a way, that $\lim_{\varepsilon \to 0} \lambda = \infty$ and $\lim_{\varepsilon \to 0} \lambda \sqrt{\varepsilon} = 0$, then $\mathcal{F}_\varepsilon \xrightarrow{\Gamma} \mathcal{F}_0$, where

$$\mathcal{F}_0(u) = \begin{cases} C\mathrm{per}_\Omega(\mathcal{I}) & u \in BV(\Omega, \{-1,1\}) \text{ and } \fint_{B_\delta(p)} u(\xi)\, \mathrm{d}m(\xi) = 0 \ \forall p \in \mathcal{P} \\ \infty & \text{else} \end{cases}$$

3.3 SDFs and the Eikonal equation

In the previous chapter we have successfully found a way of solving the surface reconstruction problem by learning an implicit representation of the underlying surface through an occupancy function. But in the beginning we said that we would actually prefer the representation through a signed distance function. So now we want to find a way of using the functionals $\mathcal{F}_\varepsilon$ and the corresponding minimizer from before to compute a signed distance function.
Therefore we have to adjust the one left degree of freedom that plays into the construction of $\mathcal{F}_\varepsilon$, and that is the double well potential. For the previous theorems to work, any suitable double well potential would work, but for the next computations, we need to explicitly state a function. So from now on let

$$W(x) := x^2 - |2x| + 1 \tag{6}$$

We can see that this is actually a double well potential. Let u_ε be a minimizer of $\mathcal{F}_\varepsilon$, that we obtain using exactly this double-well potential. We already know that these converge to a suitable occupancy function, so now we want to apply a transformation map to these minimizer and see what happens. Therefore we choose the so called logarithm transform.

Definition 3.4:
Let $\varepsilon > 0$. For the minimiser u_ε of $\mathcal{F}_\varepsilon$ we define its **log-transform** by

$$w_\varepsilon(x) = \begin{cases} -\varepsilon \log(1 - u_\varepsilon(x)) & u_\varepsilon(x) \geq 0 \\ \varepsilon \log(1 + u_\varepsilon(x)) & u_\varepsilon(x) < 0 \end{cases}$$

We see that the log-transform is continuous and does not change the sign of the original function. The goal is to show, that w_ε will now give us an approximation of a SDF. To proof this, we will not use the variational problem $\min \mathcal{F}_\varepsilon$ directly, but rather show it through the partial differential equation, that characterizes their minima.

Lemma 3.5:
Let $Q \subseteq \Omega - \bigcup_{p\in\mathcal{P}} B_\delta(p)$, with $u_\varepsilon(x) \neq 0$ for all $x \in Q$. Then on Q the function u_ε is smooth and fulfills the following partial differential equation

$$-\varepsilon^2 \Delta u_\varepsilon + u_\varepsilon - \text{sign}(u_\varepsilon) = 0.$$

Proof: Note, that since we are away from the original surface, W is differentiable with $W'(s) = 2s - 2\text{sign}(s)$. We apply theorem (2.9) with $F(\alpha, \beta, \gamma) = \frac{1}{\varepsilon} W(\beta) + \varepsilon\|\gamma\|^2$ and obtain

$$\begin{aligned} 0 &= \frac{\partial F}{\partial \beta} - \nabla \cdot \left(\frac{\partial F}{\partial \gamma}\right) \\ &= \frac{1}{\varepsilon}\left(2\beta - 2\text{sign}(\beta)\right) - \varepsilon 2\nabla \cdot \gamma \\ \Leftrightarrow \qquad &= -\varepsilon^2 \nabla \cdot \nabla u_\varepsilon + u_\varepsilon - \text{sign}(u_\varepsilon) \end{aligned}$$

□

We can see, that as ε tends to zero, the part containing the Laplacian vanishes and we are left with $u_\varepsilon = \text{sgn}(u_\varepsilon)$, which was exactly the occupancy property.

Theorem 3.6:
Let Q be the domain as in Lemma (3.5), then on Q the log-transform w_ε satisfies

$$-\varepsilon \Delta w_\varepsilon + \text{sign}(u_\varepsilon)\left(\|\nabla w_\varepsilon\|^2 - 1\right) = 0$$

Proof: To shorten notation, we write $w = w_\varepsilon$ and $u = u_\varepsilon$ and the log-transform as $w = \mp \log(1 \mp u)$. Calculating the derivatives yields

$$\begin{aligned} \frac{\partial w}{\partial x_i} &= \varepsilon \frac{1}{1 \mp u} \frac{\partial u}{\partial x_i} \\ \frac{\partial^2 w}{\partial^2 x_i} &= \pm\varepsilon \frac{1}{(1 \mp u)^2} \left(\frac{\partial u}{\partial x_i}\right)^2 + \varepsilon \frac{1}{1 \mp u} \frac{\partial^2 u}{\partial^2 x_i} \end{aligned}$$

Now from this we conclude

$$\begin{aligned}-\varepsilon\Delta w_\varepsilon \pm \left(\|\nabla w_\varepsilon\|^2 - 1\right) &= \mp\frac{\varepsilon^2}{(1\mp u)^2}\|\nabla u\|^2 - \frac{\varepsilon^2}{1\mp u}\Delta u \pm \frac{\varepsilon}{(1\mp u)^2} \mp 1 \\ &= -\frac{\varepsilon^2}{1\mp u}\Delta u \mp 1 \\ &= \frac{1}{1\mp u}\underbrace{\left(-\varepsilon^2\Delta u + u \mp 1\right)}_{=0, \text{ because of } 3.5}.\end{aligned}$$

□

Now we want to consider again, what happens, if ε goes to zero. Again, the part with the Laplacian vanishes, and we are left with the following partial differential equation, that is called the **Eikonal equation**

$$\|\nabla w\| = 1$$

We know, that w is zero only on the reconstructed surface and by the Eikonal equation, we conclude, that if we go from the surface into one direction, its value increases with constant speed and if we go into the other direction it has to decrease with constant speed. But this is equivalent to say, that w is a signed distance function of the original surface.
If we recall the assumptions of the last theorem, we needed to restrict our original domain to be away from the point of the point cloud so that we can neglect the $\int_{B_\delta(p)} u$ part of the loss functional. This means, we do not actually know if w fulfills the Eikonal equation around these points or not. So the idea is, that we modify the loss functional in such a way, that enforces the Eikonal equation to hold around all points. Therefore, let $\mu > 0$ be a constant measuring the effect of the Eikonal term. Then we add to the loss functional the current expected error, that arises from the Eikonal equation not being hold. This final loss is called PHASE-loss.

PHASE-loss

$$\min_{u\in H^1(\Omega,[-1,1])} \mathcal{F}_\varepsilon(u) + \frac{\mu}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} \|1 - \|\nabla w(p)\|^2\|$$

Note that in the PHASE loss w is the computed log-transform of u and the loss contains information on both. So in a way, this loss combines features of occupancy functions as well as signed distance functions and we get approximating functions for both of these by minimizing a single loss functional.
If the original point cloud in addition to the coordinates also contains the normal vectors in the points as features, we can rewrite the PHASE-loss as

$$\min_{u\in H^1(\Omega,[-1,1])} \mathcal{F}_\varepsilon(u) + \frac{\mu}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} \|n(p) - \nabla w(p)\|.$$

3.4 Ambrosio Tortorelli phase fields

4 Numerical implementations

4.1 Implementing PHASE loss

We want to look at the numerical experiments of the PHASE loss. We do this by using the `pytorch` environment from `python`. The network we use is a multi-layer perceptron with three hidden layers of each 128 neurons, with ReLU acitvation functions. In the output layer we use tanh as activation function, since this makes sure, the range of the network is in $[-1, 1]$. The weights are randomly initialized and adjusted by using `torch.optim.Adam`. We start by using a learning rate of 0.01 at let it decrease by a factor of 0.1 automatically, if after 1500 training epochs, there was no improvement in the total loss, using `torch.optim.lr_schedule.ReduceLROnPlateau`. We start by using the PHASE-loss functional for the training. Therefore, we need to implement

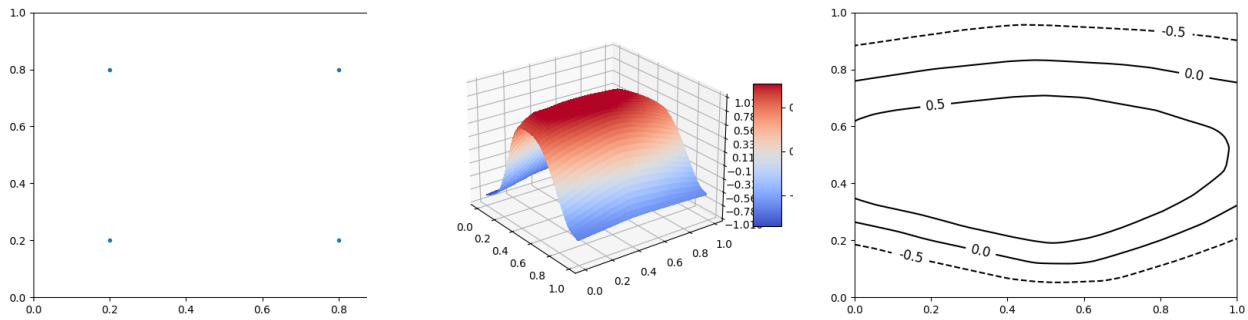
$$\varepsilon^{-\frac{1}{2}} \left(\int_\Omega W(u(\xi)) + \varepsilon \|\nabla u(\xi)\|^2 \, \mathrm{d}m(\xi) + \frac{C\varepsilon^{\frac{1}{3}}}{|X|} \sum_{x \in X} \left| \fint_{B_\delta(x)} u(\xi) \, \mathrm{d}m(\xi) \right| \right) + \frac{\mu}{|X|} \sum_{x \in X} |1 - \|\nabla w_\varepsilon(x)\||^2$$

We estimate this integral by using Monte-Carlo integration with 200 samples for the integral over Ω and 50 sample per integral over $B_\delta(x)$. Instead over taking values in a ball with radius δ, we sample points normal distributed with expectancy $\mu = x$ and variance $\sigma = 0.001$. The double well potential is the one in (6). For the other parameters we start by choosing $\varepsilon = 0.01$, $C = 14$ and $\mu = 0.1$. We will later see the effects of increasing/decreasing the different parameters. If we take the gradient of the log-transform in on of the point of the point cloud, i.e. where $u_\varepsilon(x) = 0$ we obtain

$$\nabla w_\varepsilon(x) = \nabla \left(\mp \sqrt{\varepsilon} \log(1 \pm u_\varepsilon(x)) \right) = \mp \sqrt{\varepsilon} \frac{\mp \nabla u_\varepsilon(x)}{1 \mp u_\varepsilon(x)} = \sqrt{\varepsilon} \nabla u_\varepsilon(x).$$

That means we can calculate the gradient of the log-transform without actually computing the log-transform. The gradients of the network function are automatically computed by using `torch.autograd.grad`.

Now we want to take a look at some experimental results. To start, we sample points from a square.

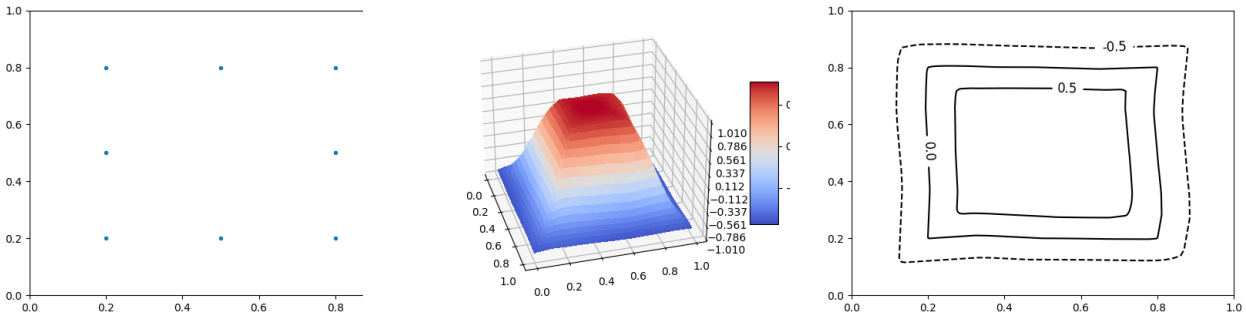


(a) Original point cloud (b) 3D graph (c) Contour plot

Figure 8: Reconstructing a square from four points

We see that the reconstruction does not work, since for every point the distance to the boundary

is shorter, than to the next point. Therefore the total gradient gets minimized by connecting each two points with two straight lines. In order to get a square, we need to increase the number of points.



(a) Original point cloud (b) 3D graph (c) Contour plot

Figure 9: Reconstructing a square from 8 points

As hoped, by increasing the number of points we can see that the square was successfully reconstructed. Even though the lines are not perfectly straight, we can clearly see the shape of the square. However, the plot that we can see in (9) is not the only result that we get, while training the network. Occasionally, the network will converge to the following function, that is mostly ± 1 and has spikes around the points of the point cloud, that ensure that the integrals over the δ-balls get minimized. Considering the gradients of the network parameters in the end, we can see, that this also is a local minimum.

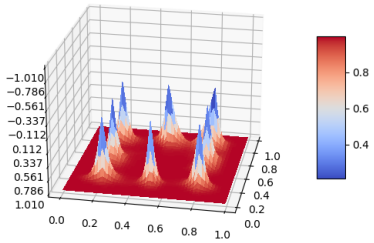
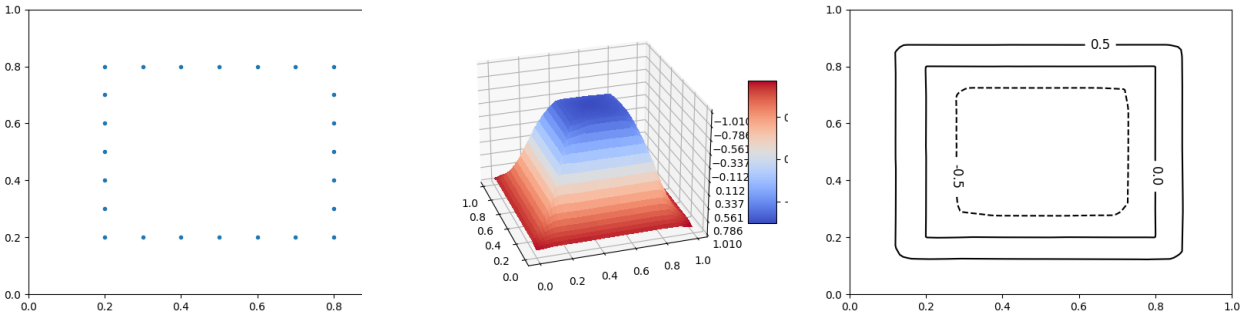


Figure 10: Alternative local minimum for the PHASE-loss

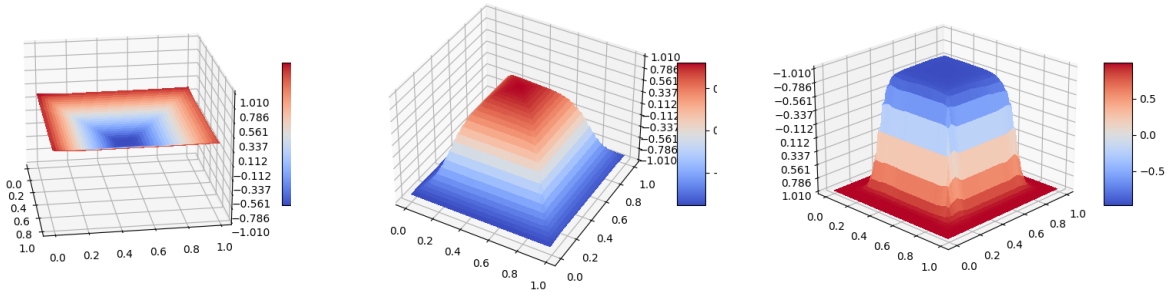
There are two possible ways, of how we could get rid of this. One possibility is not to take randomly initialized weights, but ones that position the network function „close“ to the occupancy function, and therefore ensuring convergence.
The other thing we could to, is again increase the total number of points.



(a) Original point cloud (b) 3D graph (c) Contour plot

Figure 11: Reconstructing a square from 24 points

Now we can see a big improvement in the contour plot. The reconstruction of the square is nearly perfect and we no longer get a convergence to any other local minimum. Something else that we especially in the contour lines of $\pm\frac{1}{2}$, is that the corners of the square are slightly rounded. This is due to the fact, that our solution is smooth (see), but for a perfect square, we would get that the occupancy function can not be differentiable around the medial axis, i.e. the lines that go from the center through the corners. That means, this is really the best we can expect. Now we want to look the effect of the different parameters.

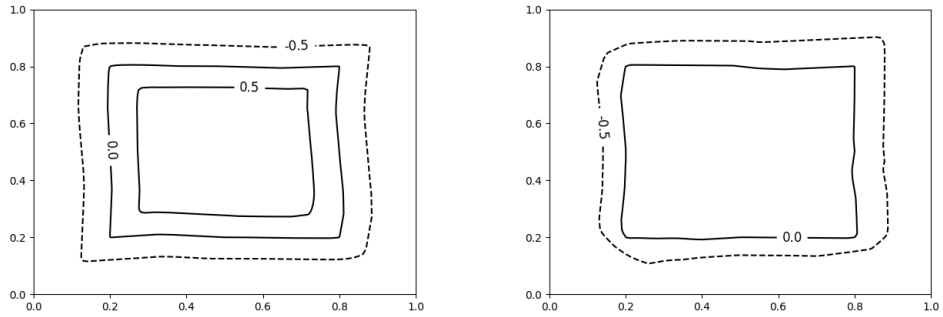


(a) $\varepsilon = 0.1$ (b) $\varepsilon = 0.01$ (c) $\varepsilon = 0.001$

Figure 12: Reconstructing a square from 24 points with varying ε parameter

We can see, that as suggested, the more we decrease the ε parameter, the more the resulting function converges to a proper occupancy function. If ε is too big, the occupancy property is nowhere hold, since the $\int W(u)$ part in the integral does not contribute much and the function converges to 0.

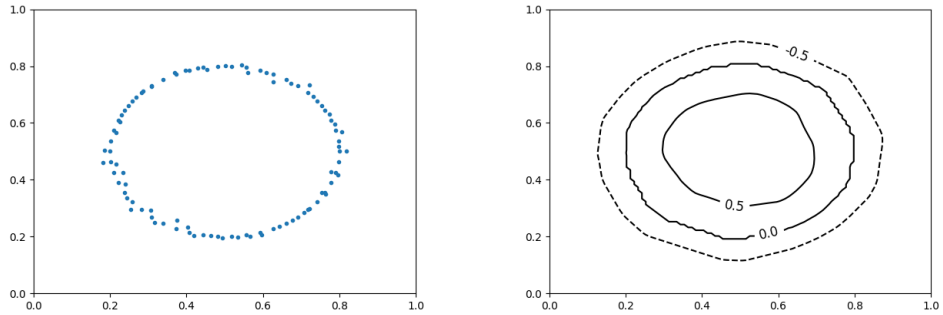
If we change the μ parameter on the point cloud with 24 points, we will not see any difference. To really see an effect of the Eikonal term, we have to go back to the smaller point cloud.



(a) $\mu = 0.1$ (b) $\mu = 0$

Figure 13: Reconstructing a square from 8 points with varying μ parameter

Here, we can see a slightly improvement of the contour, if we choose $\mu > 0$, especially around the corner points. It also prohibits the function to be close to zero in the interior, but really enforces the phase field property there.
Next up we want to see, how the algorithm handles noise. Therefore we sample 30 equidistant point from a circle with radius 0.3. To every second point we add noise, that is uniformly distributed in $[-0.02, 0.02]^2$.



(a) Original point cloud (b) Contour plot

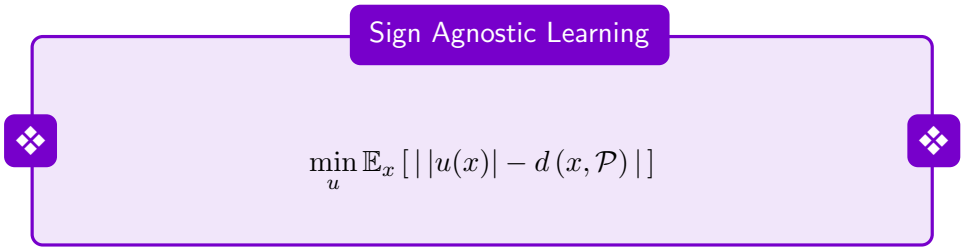
Figure 14: Effect of noise

We can see, that the algorithm handles noise quiet well and we can still clearly recognize the circle in the end.

4.2 Improving the network

Before we continue to the Ambrosio-Tortorelli phase fields, we want to make some adjustments on the network.
The first thing, we want to take care of, is the convergence of the network. We have seen in (10), that multiple local minima to the PHASE-loss functional exist, but only one of them is desirable for our purposes. To ensure, that we converge to the right one, we need to recall, how the process of minimization works. The 'Adam' optimization algorithm we use, belongs to the class of gradient descent algorithm. Optimizing a function with using these algorithms always works after the same scheme: We start in a starting point, that may be randomly generated. In our case, a 'point' is some vector, representing the different weights in the network. From

there on, we move into the negative direction of the gradient, where the step size equals the learning rate, until we are sufficiently close to the minimum. This means, if the functional we minimize has multiple local minimal, the algorithm will converge to the one, that lies closest to the starting point.
Since we know the surface reconstruction problem has multiple local minima our goal has to be to find initials weights, that are closer to the occupancy solution, than to the one in (10).
To do this, we want a new approach to the surface reconstruction problem from [AL19]. In this paper, they learn a signed distance function as the minimizer of the following functional.



The idea here is that $|u|$ approximates the unsigned distance to the underlying point cloud $\mathcal{P}$ in terms of the metric d. If we choose d to be the discrete metric, a proper occupancy function would be a solution. If we choose the metric to be the standard Euclidean distance, the signed distance function would be a solution. The loss functional can be improved, by also comparing the unsigned gradients of the network function and the distance function ([**?**]).
The problem is however, that again there are multiple local minima to this problem. Once we have the desired signed distance function as a solution, but also an unsigned distance function, where u does not change sign along the original points can be a solution to it. In order to prevent the wrong convergence, the authors use a so called **geometric initialization**, which is a proper initialization of the networks parameters. It is defined as follows: Initialize all weights, that are associated with hidden layers and do not come from bias terms, randomly from a normal distribution with mean value 0 and variance $\sqrt{\frac{2}{d_i}}$. Here d_i is the number of neurons in the next layer. Set all the corresponding bias parameters to zero. In the final output layer, set the weights to $\sqrt{\frac{\pi}{d}}$, with d being the output dimension of the neural network and the final bias term to $-r$ for some $r > 0$. If we do this, we obtain the following theorem (Theorem 1 in [AL19]):

> **Theorem 4.1**:
> Suppose we sample a point cloud from a sphere with radius r. Using the geometric initialization as described above, the neuronal network will converge to an approximation of the signed distance function solution.

This proof might only yield convergence for sphere-shaped objects, while in reality we get good results on all kinds of data input. Now we want to get back to the PHASE-loss. Even though, this loss is very different from the SAL-loss, the geometric initialization still provides a starting point for the gradient descent, that is close to our desired function. We can observe, that the network now very rarely converges to a spike-like solution as in (10).

Now we want to take care of another problem. We want our network also to handle larger point clouds and more complicated objects. In order for this to work we need to increase the

number of neurons and layers in our network to expect good results. However, this extension comes with a price, called **vanishing gradient problem**. This explanation is based on [**?**]. In order to understand what the problem is, we need to revisit how the training of the network works, in particular, how one step of the gradient descent algorithm works. After evaluating the loss functional, we need a way to adjust the weights of the neural network. This is done using **backpropagation**. It means we calculate the gradient in the current error of the loss functional with respect to the weights. This shows how much to the error the weights w of the last hidden layer contribute. Based on that, we take adjustments

$$w \leftarrow w - h\frac{\partial L}{\partial w}.$$

Here, h is the learning rate. To update the weights in the previous layers of network, we use the chain rule for differentiation iterative. This means for example if we consider a single weight w in the i-th layer of a neuronal network with a total of l layers, we compute the gradient of the loss functional with respect to w as the product of $l - i$ gradients of the loss functional with respect to the previous weights. In practice, most of the gradients that appear and also the learning rate will have absolute values much smaller than 1. So, if we have a high number of layers in the network, the gradient in the early layers, will be close to zero and therefore vanish. This means, that during the training of the network, the parameters of these layers will not update at all and therefore we can conclude, that just increasing the layer sizes will not yield in a better learning. In order to deal with this problem, we need to build alternative paths for the gradients to directly be connected to the early layers, without passing through all layers. The idea is that from the input layer, we add an additional connection to the rear layers, that is just the identity. So, we concatenate the feature vector with the original input. These connections are called **skipping layers**. This allows us to backpropagate the error into the early layers, without passing through the whole network and therefore prohibits the vanishing gradient problem.

4.3 Implementing Ambrosio Tortorelli loss

Index

List of Figures

References

[Ada20] Adaloglou, Nikolas: *Intuitive Explanation of Skip Connections in Deep Learning*. https://theaisummer.com/skip-connections/. Version: 2020

[Agg18] Aggarwal, Charu C.: *Neural Networks and Deep Learning*. Springer International Publishing, 2018 https://www.ebook.de/de/product/33161337/charu_c_aggarwal_neural_networks_and_deep_learning.html. – ISBN 3319944622

[AL19] Atzmon, Matan ; Lipman, Yaron: SAL: Sign Agnostic Learning of Shapes from Raw Data. (2019), November

[AL20] Atzmon, Matan ; Lipman, Yaron: SALD: Sign Agnostic Learning with Derivatives. (2020), Juni

[Arn20] Arnold, Anton: *Variationsrechnung*. Lecture notes, Juli 2020

[CBC+01] Carr, J. C. ; Beatson, R. K. ; Cherrie, J. B. ; Mitchell, T. J. ; Fright, W. R. ; McCallum, B. C. ; Evans, T. R.: Reconstruction and representation of 3D objects with radial basis functions. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*, ACM Press, 2001

[Del34] Delaunay, Boris N.: Sur la sphère vide. In: *Bulletin of Academy of Sciences of the USSR. 7* (1934), Nr. 6, S. 793–800

[Gio75] Giorgi, E. D.: Sulla convergenza di alcune successioni d'integrali del tipo dell'area. In: *Collectionof articles dedicated to Mauro Picone on the occasion of his ninetieth birthday*, 1975, S. 277–294

[Giu84] Giusti, Enrico: *Minimal surfaces and functions of bounded variation*. Boston : Birkhäuser, 1984. – ISBN 3764331534

[Han19] Hanin, Boris: Universal Function Approximation by Deep Neural Nets with Bounded Width and ReLU Activations. In: *Mathematics* 7 (2019), oct, Nr. 10, S. 992. http://dx.doi.org/10.3390/math7100992. – DOI 10.3390/math7100992

[HQD02] Huong Quynh Dinh, Greg S. Greg Turk T. Greg Turk: Reconstructing Surfaces By Volumetric Regularization Using Radial Basis Functions. In: *IEEE Trans. on Pattern Analysis and Machine Intelligence 24*, 2002, S. 1358–1371

[KH13] Kazhdan, Michael ; Hoppe, Hugues: Screened poisson surface reconstruction. In: *ACM Transactions on Graphics* 32 (2013), jun, Nr. 3, S. 1–13. http://dx.doi.org/10.1145/2487228.2487237. – DOI 10.1145/2487228.2487237

[Lip21] Lipman, Yaron: Phase Transitions, Distance Functions, and Implicit Neural Representations. In: *ICML 2021* (2021), Juni

[MK06] Michael Kazhdan, Hugues H. Matthew Bolitho B. Matthew Bolitho: Poisson surface reconstruction. In: *Eurographics Symposium on Geometry processing* (2006), S. 61–70

[Mut10] Muthukumar, T.: Introduction to Γ-convergence. In: *Workshop on Multiscale Analysis and Homogenization, IISc. Bangalore*. Department of Mathematics & Statistics, Indian Institute of Technology-Kanpur, Kanpur, Uttarpradesh, India, 208016, Juli 2010

[PA07] P. Alliez, Y. Tong M. D. D. Cohen-Steiner: Voronoi-based Variational Reconstructionof Unoriented Point Sets. In: *Computer Graphics Forum (Proc. of the Symposium on Geometry Processing)*, 2007

[Pou20] Poux, Florent: *How to represent 3D data?* https://towardsdatascience.com/how-to-represent-3d-data-66a0f6376afb. Version: Mai 2020. – Last Access: 24.11.2021

[Run01] Runge, C.: Über empirische Funktionen und die Interpolation zwischen äuidistanten Ordinaten. In: *Zeitschrift für Mathematik und Physik* Bd. 46, 1901, S. 224–243

[TO02] Turk, Greg ; O'Brien, James F.: Modelling with implicit surfaces that interpolate. In: *ACM Transactions on Graphics 21* Bd. 4, ACM Press, 2002, S. 855–873

[ZO02] Zhao, Hongkai ; Osher, Stanley: Visualization, analysis and shape reconstruction of unorganized data sets. In: *Geometric Level Set Methods in Imaging, Vision and Graphics* (2002)

[ZZW19] Zhong, Deyun ; Zhang, Ju ; Wang, Liguan: Fast Implicit Surface Reconstruction for the Radial Basis Functions Interpolant. In: *Applied Sciences* 9 (2019), dec, Nr. 24, S. 5335. http://dx.doi.org/10.3390/app9245335. – DOI 10.3390/app9245335